OpenNF: Enabling Innovation in Network Function Control

Aditya Akella

With: Aaron Gember, Raajay Vishwanathan, Chaithan Prakash, Sourav Das, Robert Grandl, and Junaid Khalid

Network functions, or Middleboxes

Introduce custom packet processing functions into the network

Firewall

Caching Proxy

Intrusion Prevention

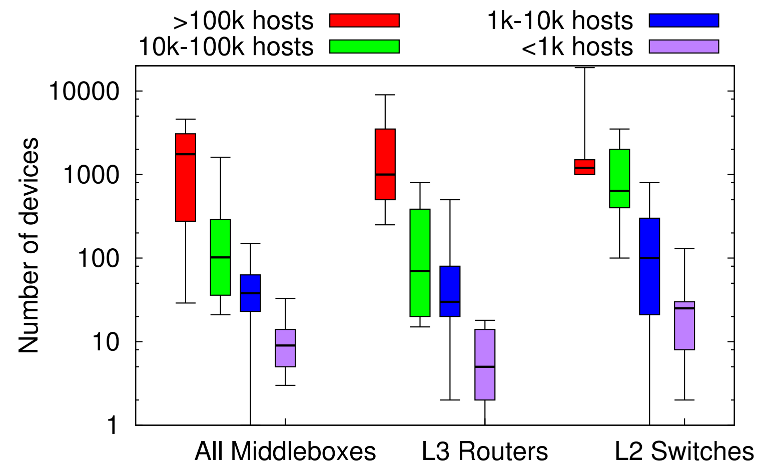
Traffic scrubber

Load balancer

SSL Gateway

WAN optimizer

...



[Sherry et al., SIGCOMM 2012]

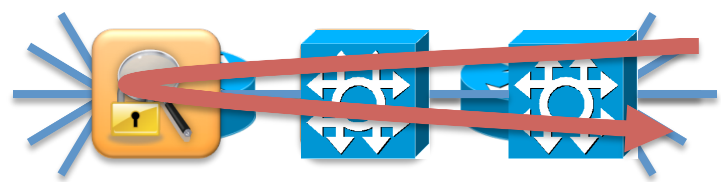
Stateful: detailed book-keeping for network flows

Common in enterprise, cellular, ISP networks

State-of-the-art

- Network functions virtualization (NFV)

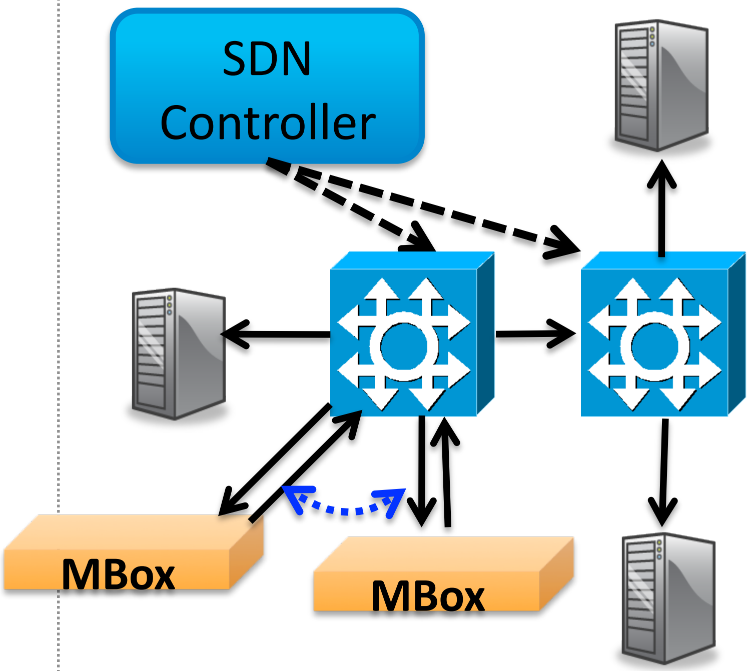
- Software-defined networking (SDN)



Distributed processing

Dynamic reallocation to coordinate processing across instances

Load balancing

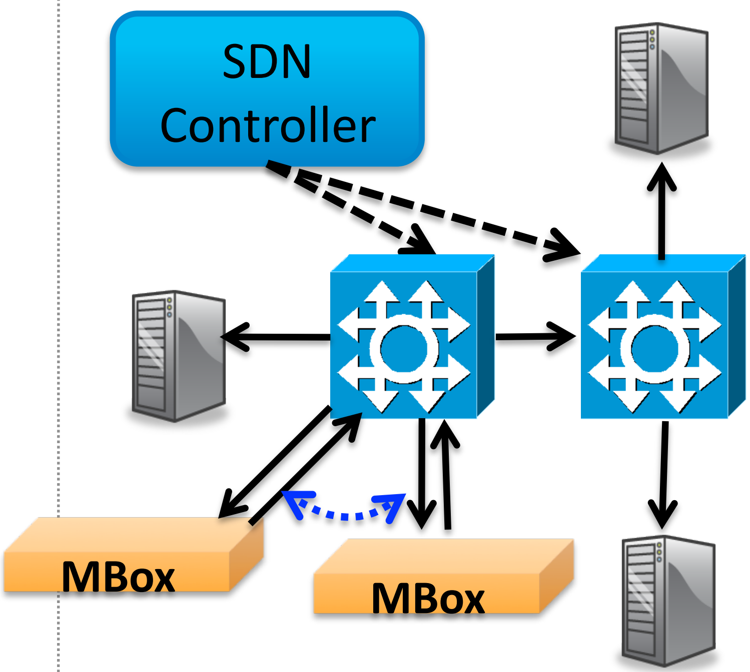


Extract maximal performance at a given $$

Distributed processing

Dynamic reallocation to coordinate processing across instances

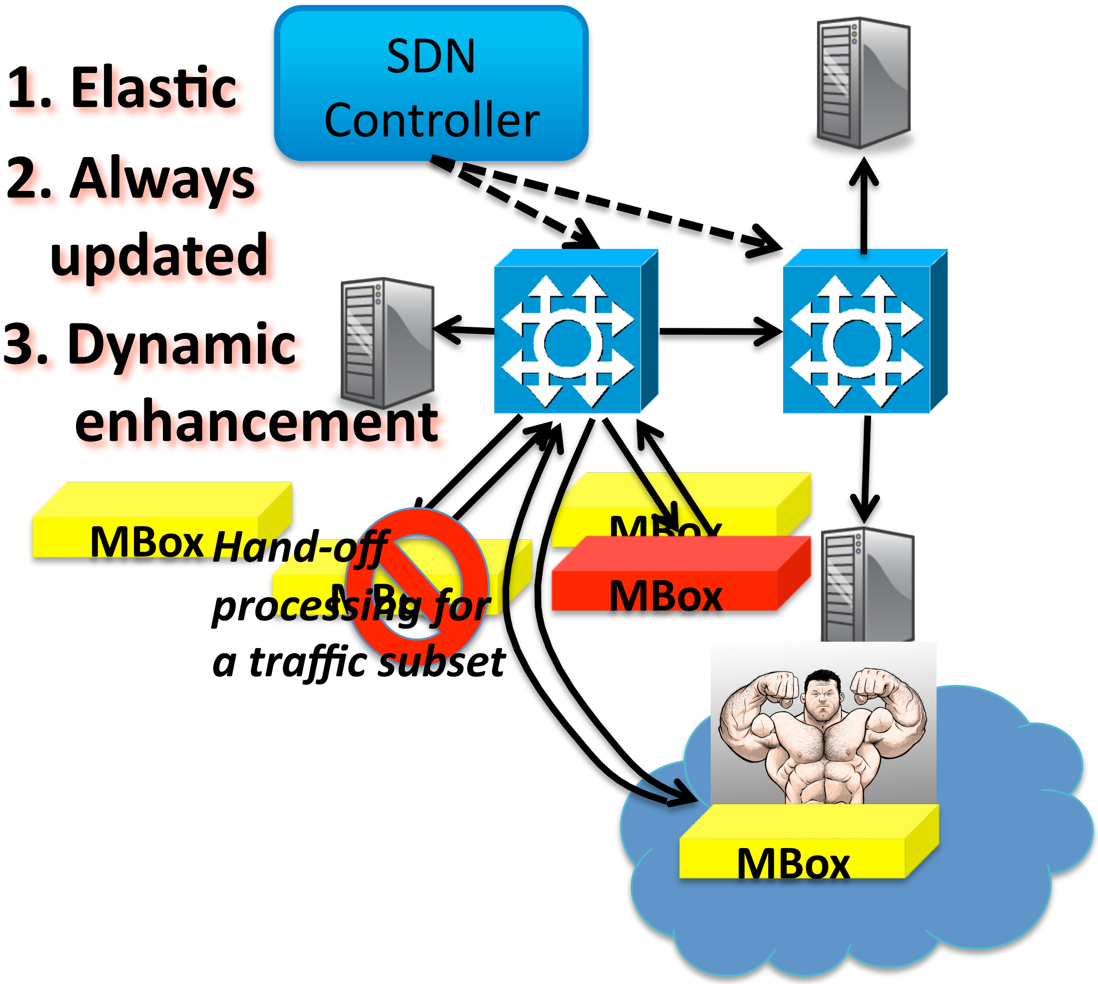
Load balancing



Extract maximal performance
at a given $$

Novel abstractions

1. Elastic
2. Always updated
3. Dynamic enhancement



What's missing today?

The ability to **simultaneously**

Meet tight SLAs

– E.g., time outdated NFs are used to process (long) flows is less than 3 seconds

Ensure safe reallocation

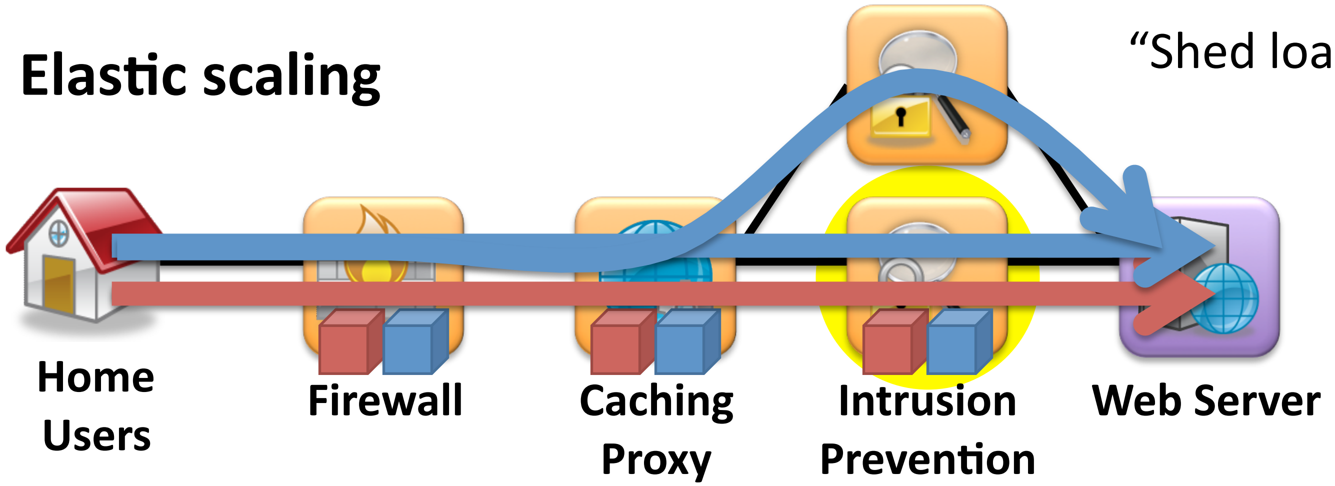
– E.g., IDS raises alerts for *all* HTTP flows containing known malware packages

Keep costs low

– E.g., shut down idle resources when not needed

Why? SDN example

Elastic scaling



Not moving flows → bottleneck persists → Responsiveness!

Naively move flows → *associated state??* → Output equiv.!
→ incorrect behavior

Need joint control over forwarding and NF state

OpenNF

- Overview and challenges
- OpenNF
  - Requirements
  - Key ideas
  - Applications
- Evaluation

OpenNF

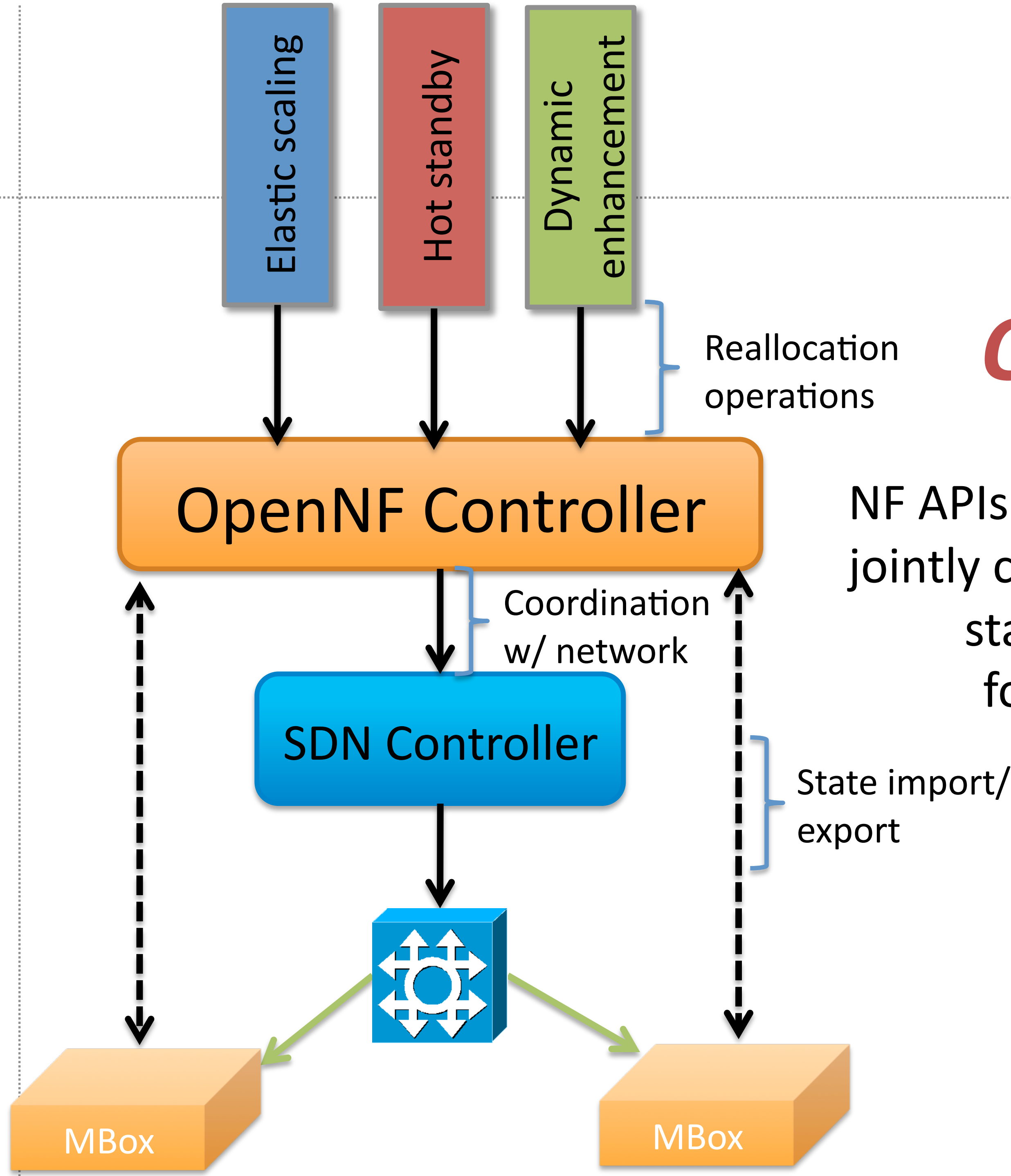
Support key semantics in distributed processing:

safe reallocation of processing at any time

Output equivalence

1. Detailed understanding of state
2. Staged updates for safe live state migration
3. App knobs to control overhead vs. performance

Any relloc. policy enforced any time, finishes "soon"



OpenNF

NF APIs and control plane for jointly controlling internal NF state and network forwarding state

Key Challenges

1: Many NFs, minimal changes

– Undesirable to force NFs to conform to certain state structures or allocation/access strategies

2: Reigning in race conditions

– Packets may arrive while state is being moved → updates *lost* or *re-ordered*; state *inconsistency*

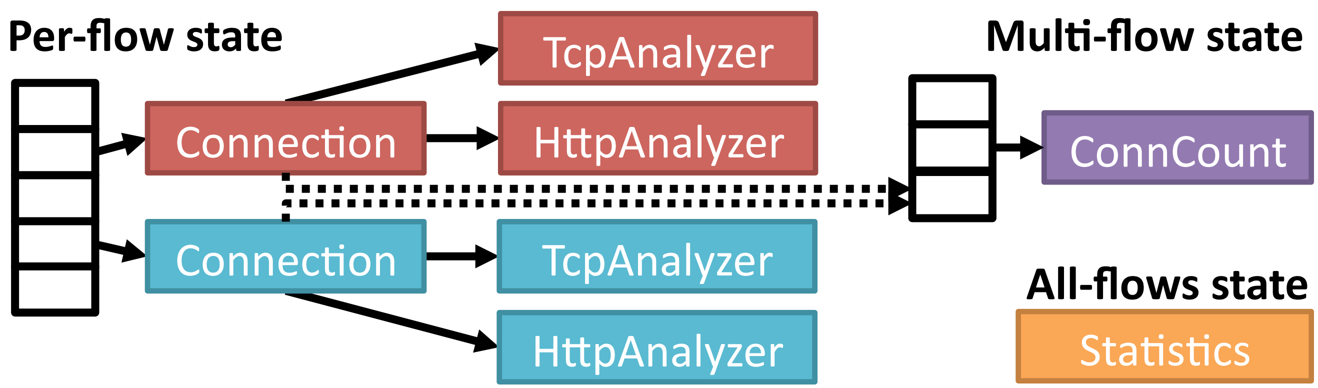
3: Bounding overhead

– State operations at different granularities

– Flexibility in choosing guarantees

NF state taxonomy

State created or updated by an NF applies to either a single flow or a collection of flows



Classify state based on **scope**

Flow provides a natural way for reasoning about which state to move, copy, or share

API to export/import state

Three simple functions: `get`, `put`, `delete(f)`

- Version for each scope (per-, multi-, all-flows)
- Filter `f` defined over packet header fields

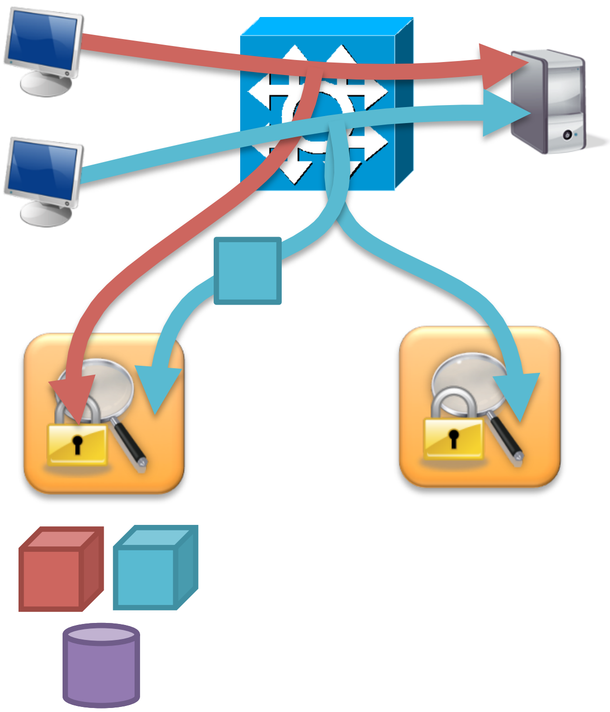
NFs responsible for

- Identifying and providing all state matching a filter
- Combining provided state with existing state

No need to expose internal state organization
No changes to conform to a specific allocation strategy

Operations

"Reallocate port 80 to NF2"

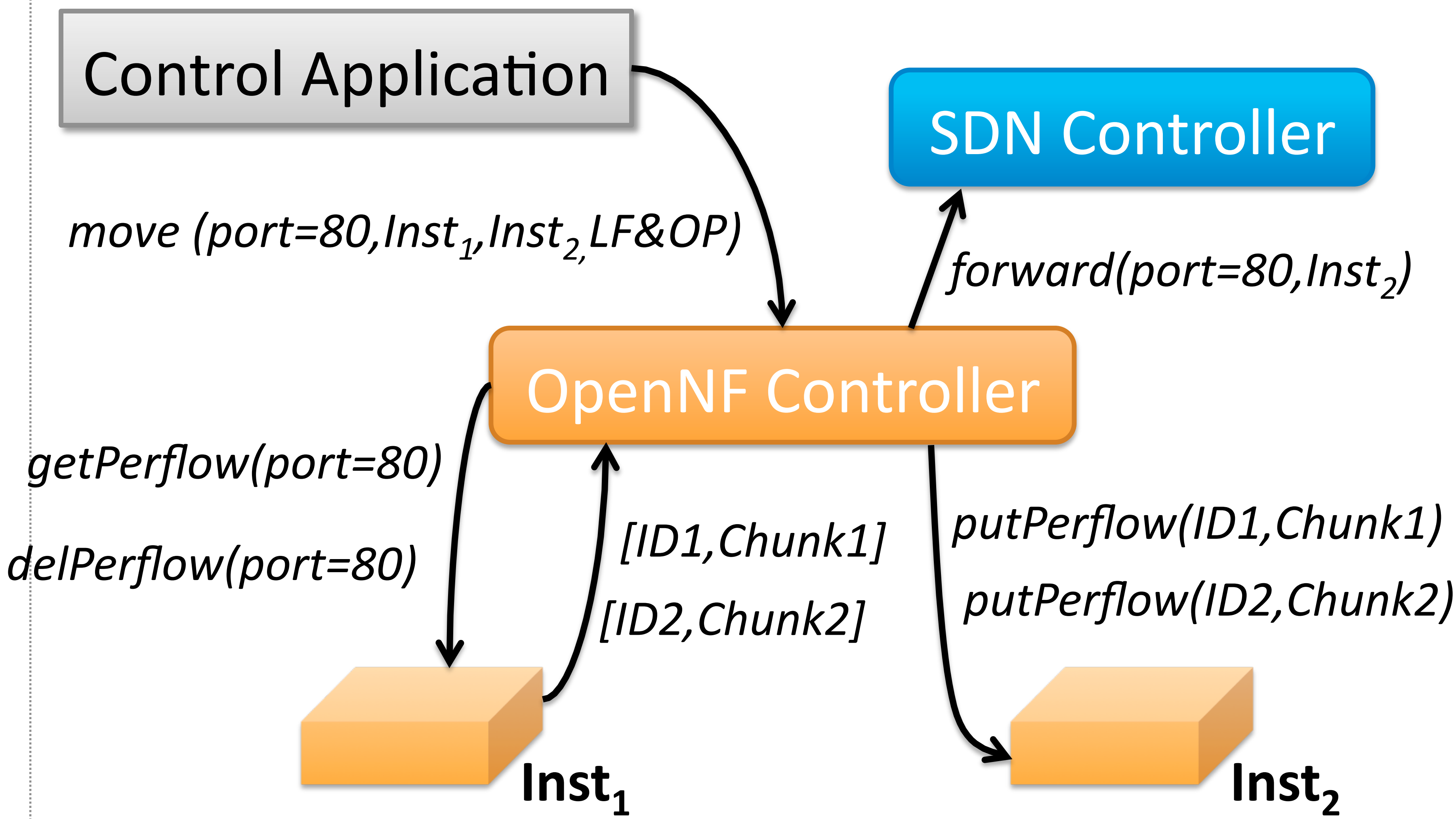


move flow-specific NF state at various granularities

copy and combine, or share, NF state pertaining to multiple flows

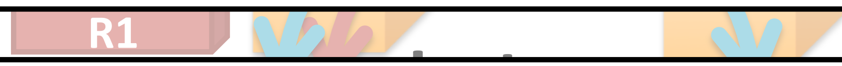
Semantics for move (loss-free, order-preserving), copy/share (various notions of consistency)

Move


Control Application
SDN Controller
move (port=80,Inst1,Inst2,LF&OP)
forward(port=80,Inst2)
OpenNF Controller
getPerflow(port=80)
delPerflow(port=80)
[ID1,Chunk1]
[ID2,Chunk2]
putPerflow(ID1,Chunk1)
putPerflow(ID2,Chunk2)
Inst1
Inst2

Lost updates during move

Loss-free: All state updates due to packet processing should be reflected in the transferred state, and all packets the switch receives should be processed

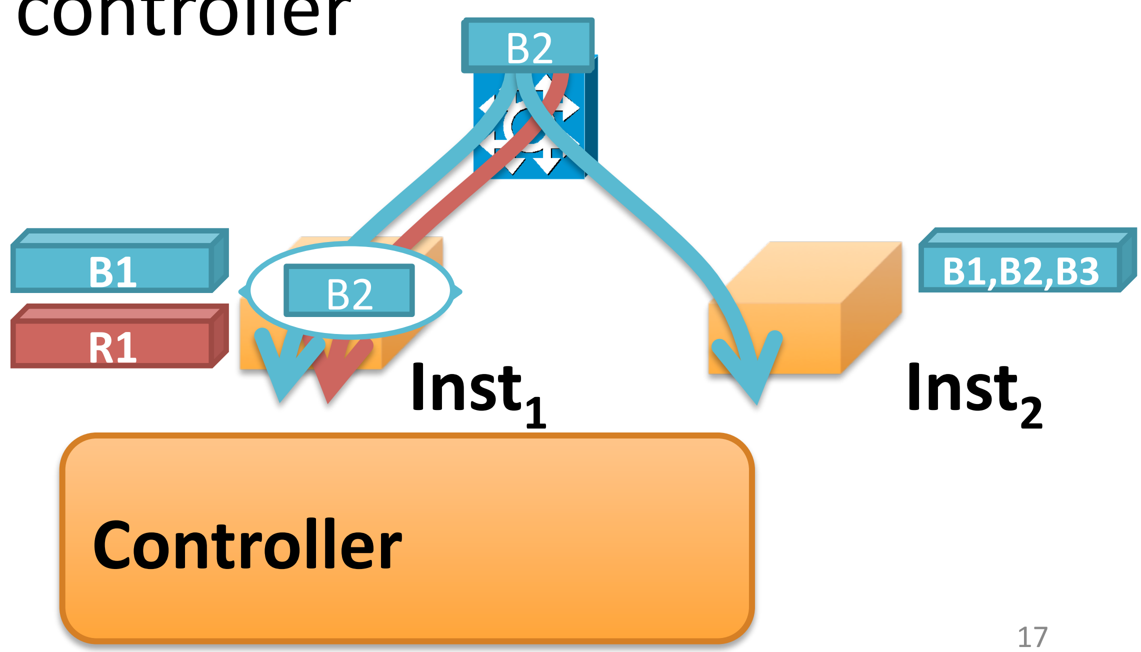


Key idea: Event abstraction to prevent, observe and sequence state updates

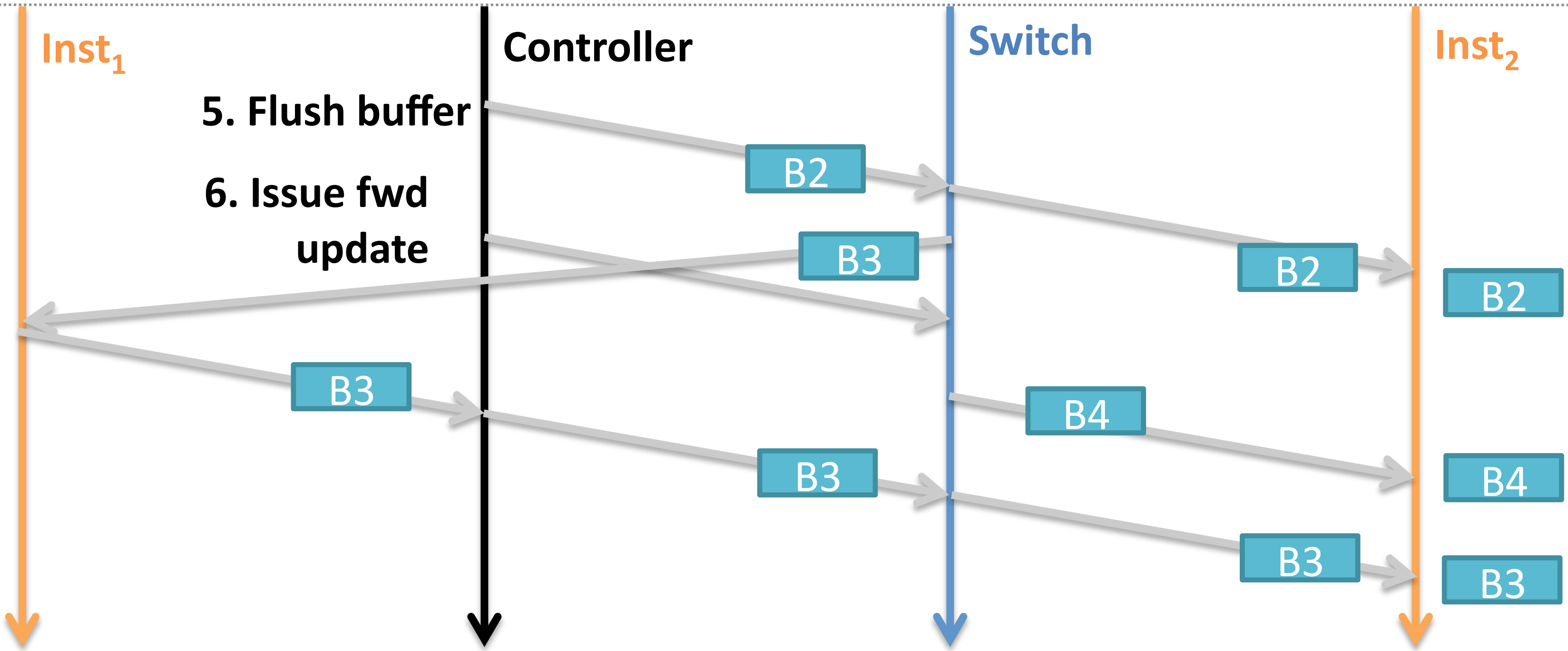
Loss-free move using events

Stop processing; buffer at controller

1. `enableEvents(blue,drop)` on Inst_1;
2. `get/delete` on Inst_1
3. Buffer events at controller
4. `put` on Inst_2
5. Flush packets in events to Inst_2
6. Update forwarding



Re-ordering of updates



Order-preserving: All packets should be processed in the order they were forwarded to the NF instances by the switch

Two-stage update to track last packet at NF1

Order-preserving move

Track last packet; sequence updates

Flush packets in events to $Inst_2$ w/ “do not buffer”

`enableEvents(blue,buffer)` on $Inst_2$

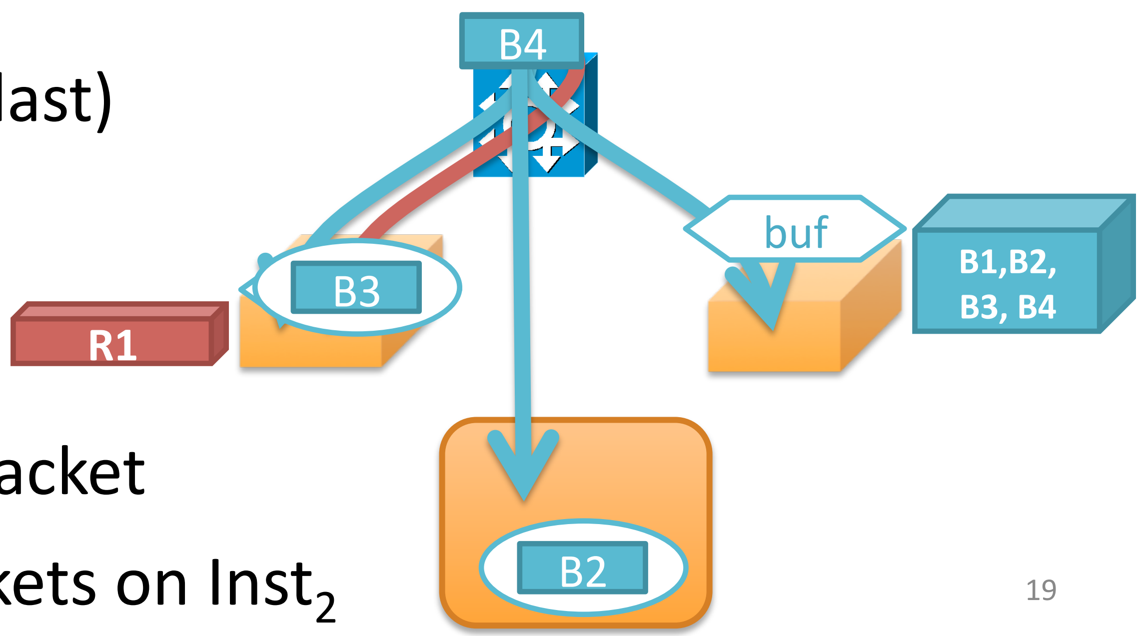
Forwarding update: send to $Inst_1$ & controller

Wait for packet from switch (remember last)

Forwarding update: send to $Inst_2$

Wait for event from $Inst_2$ for last $Inst_1$ packet

Release buffer of packets on $Inst_2$

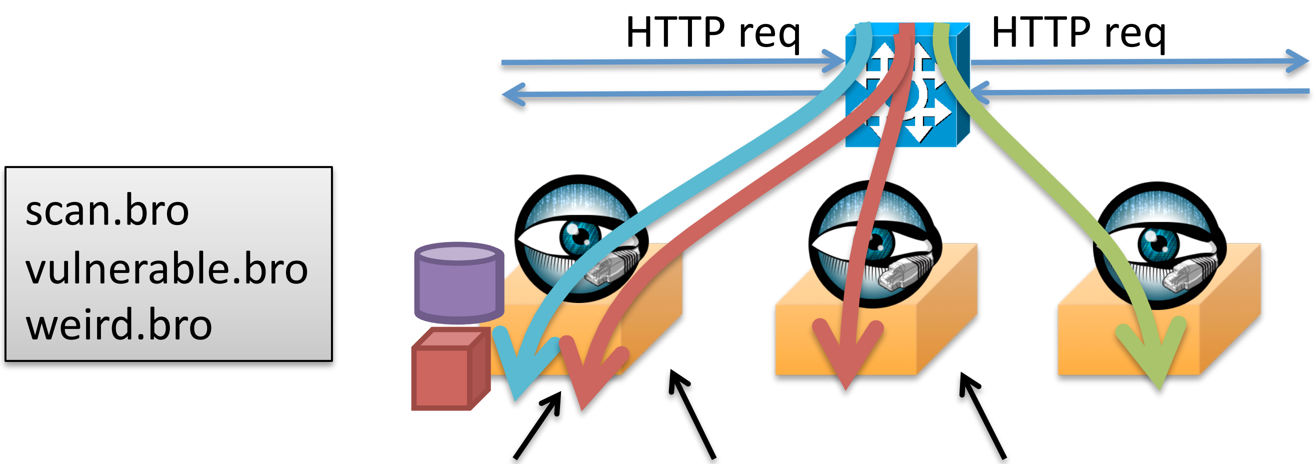


Bounded Overhead

Apps decide

- ***granularity of reallocation operations***
  - move, copy or share
  - filter, scope
- ***guarantees desired***
  - move: no-guarantee, loss-free, loss-free + order-preserving
  - copy: no or eventual consistency
  - share: strong or strict consistency

Example app: Load-balanced network monitoring




```
movePrefix(prefix,oldInst,newInst):
  copy(oldInst,newInst,{nw_src:prefix},multi)
  move(oldInst,newInst,{nw_src:prefix},per,LF+OP)
  while (true):
    sleep(60)
    copy(oldInst,newInst,{nw_src:prefix},multi)
    copy(newInst,oldInst,{nw_src:prefix},multi)
```

scan.bro

vuln.bro

weird.bro

scan.bro

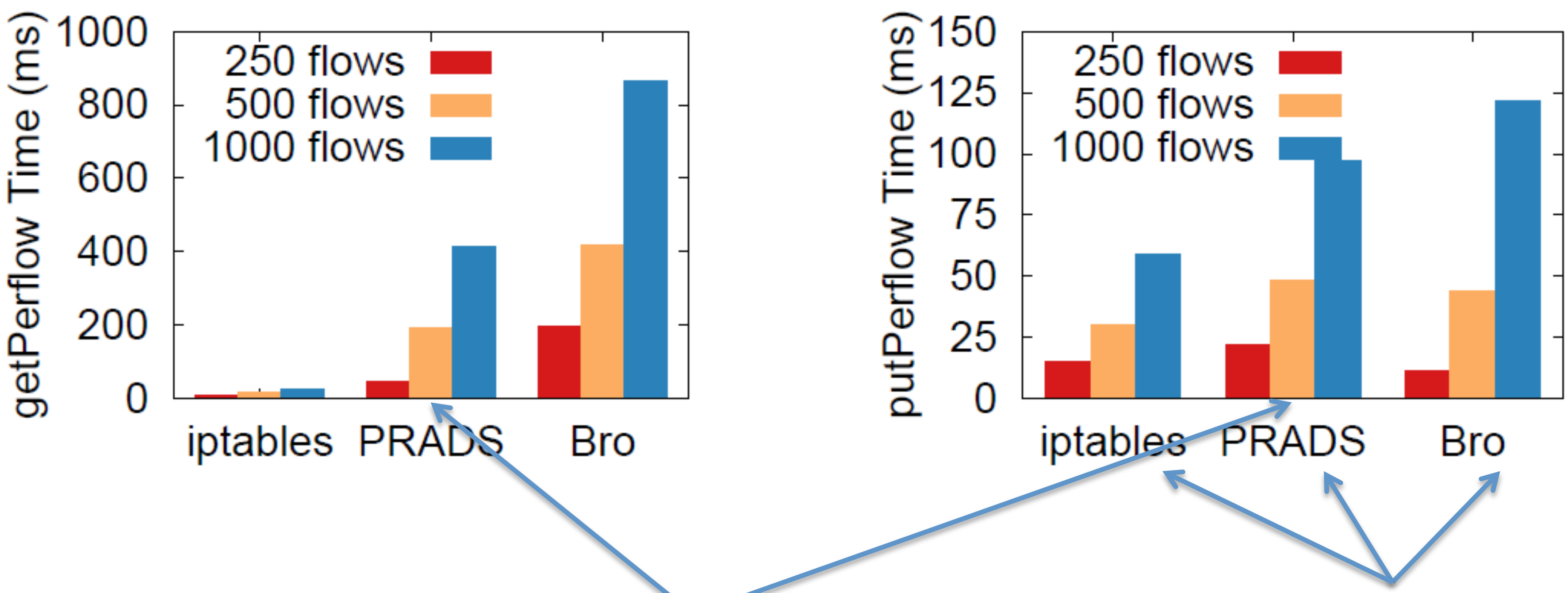
Implementation

OpenNF Controller (≈5.3K lines of Java)
- Written atop Floodlight

Shared NF library (≈3K lines of C)

Modified NFs (3-8% increase in code)
- Bro (intrusion detection)
- PRADS (service/asset detection)
- iptables (firewall and NAT)
- Squid (caching proxy)

Microbenchmarks: NFs



Serialization/deserialization costs dominate

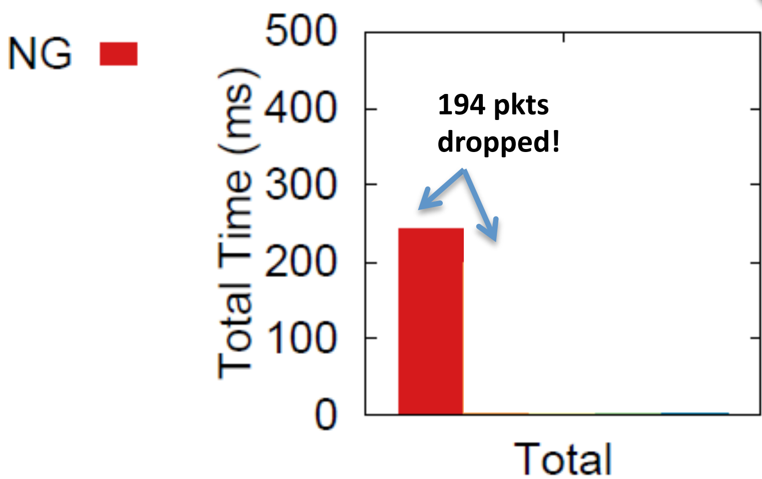
Cost grows with state complexity

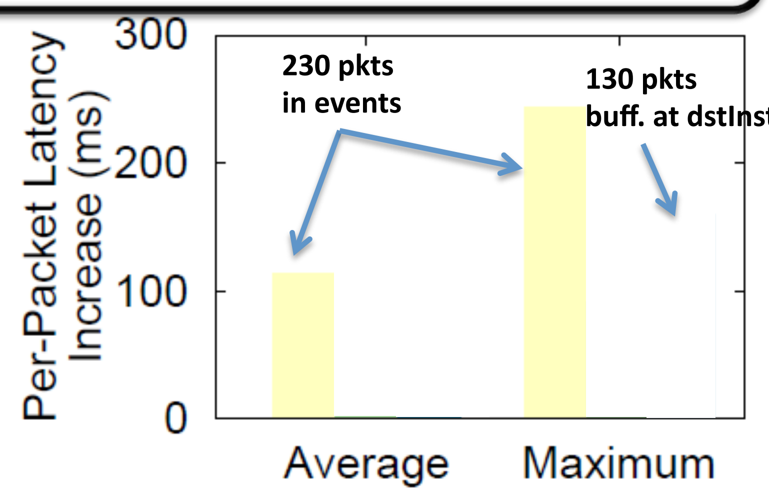
Microbenchmarks: Operations

State: 500 flows in PRADS; Worload: 1000 pkts/s; 50% util

Move: all flows w/ per-flow state

D = f(load,state,speed)





Copy (MF state) – 176ms

Share (strong) – 7ms *per* pkt

Guarantees come at a cost!

Macrobenchmarks: End-to-end benefits

Load balanced monitoring with Bro IDS
- *Load*: replay cloud trace at 10K pkts/sec
- *At 180 sec*: move HTTP flows (489) to new Bro
- *At 360 sec*: move HTTP flows back to old Bro

OpenNF scaleup: 260ms to move (optimized, loss-free)
- Log entries equivalent to using a single instance

VM replication: 3889 incorrect log entries
- Cannot support scale-down

Forwarding control only: scale down delayed by more than 1500 seconds

Wrap Up!

- OpenNF enables rich control of the packet processing happening across instances of an NF
- Key safety guarantees, efficient, overhead control, minimal NF modifications

http://opennf.cs.wisc.edu

Relation w/ SDN (research)

SDN: control over router/switch state
OpenNF: control over NF state

SDN: controller can "compute" then write state; knows how state is being used
OpenNF: limited to "handling" state

SDN (purist): dumb network elements w/o control plane
OpenNF: "not so pure"; NF-internal "control" plane??

SDN: consistency semantics an afterthought
OpenNF: semantics from the ground up

Backup

Copy and share

Used when multiple instances need to access a particular piece of state

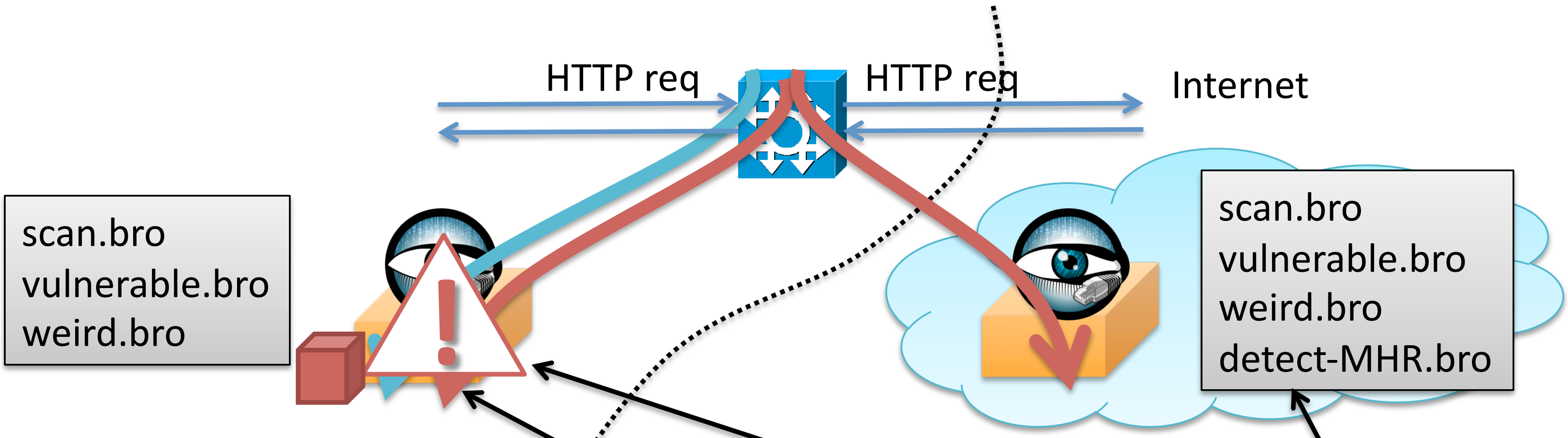
Copy – eventual consistency

- Issue once, periodically, based on events, etc.

Share – strong

- All packets reaching NF instances trigger an event
- Packets in events are released one at a time
- State is copied between packets

Example app: Selectively invoking advanced remote processing




```
enhanceProcessing(flowid,locInst):
  move(locInst,cloudInst,flowid,per,LF)
```

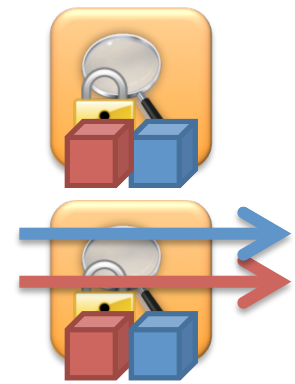
checks md5sum of HTTP reply

Enterprise n/w

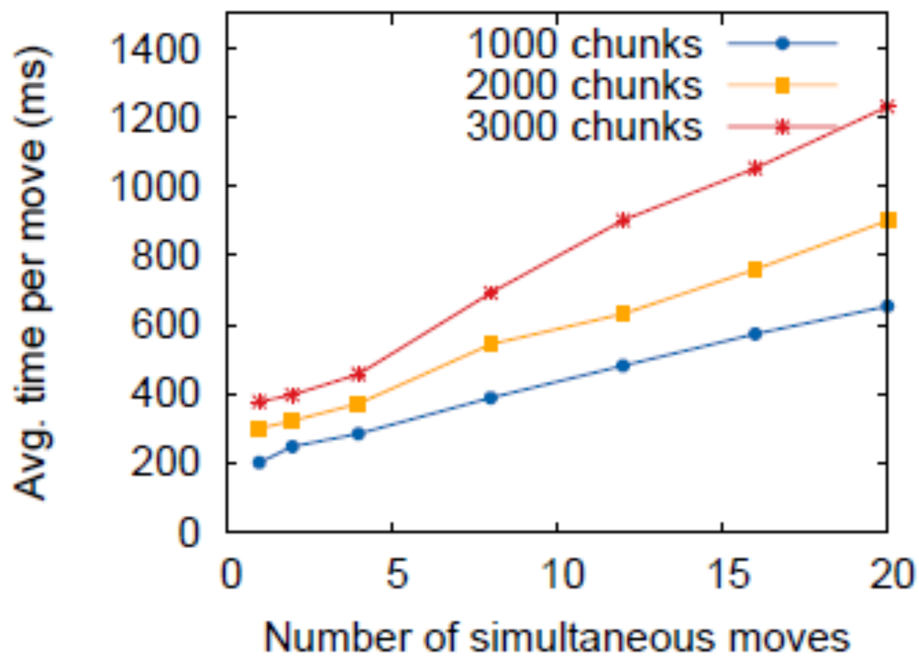
No need for:
(1) order-preservation
(2) copying multi-flow state

Existing approaches

- Control over routing (PLayer, SIMPLE, Stratos)
- Virtual machine replication
  - Unneeded state => incorrect actions
  - Cannot combine => limited rebalancing
- Split/Merge and Pico/Replication
  - Address specific problems => limited suitability
  - Require NFs to create/access state in specific ways => significant NF changes



Controller performance



Improve scalability with P2P state transfers

Macrobenchmarks: Benefits of Granular Control

Two clients make HTTP requests

- 40 unique URLs

Initially, both go to Squid1

20s later → reassign client 1 to Squid2

Metric	*Ignore*	*Copy-client*	*Copy-all*
Hits @ S1	117	117	117
Hits @ S2	*crashed*	39	50
State transferred	0	4MB	54MB

← Granularities of copy